清华师哥封神之作！RocketMQ核心笔记疯传Ali内网

原创

普通网友 于 2021-07-26 13:55:13 发布 35 收藏
分类专栏： 面试题 文章标签： java

消息队列（RocketMQ ）作为高并发系统的核心组件之一，能够帮助业务系统解构提升开发效率和系统稳定性。

RocketMQ 是一款分布式、队列模型的消息中间件，具有以下特点：

- **能够保证严格的消息顺序**
- **提供丰富的消息拉取模式**
- **高效的订阅者水平扩展能力**
- **实时的消息订阅机制**
- **亿级消息堆积能力**

选择RocketMQ的理由：

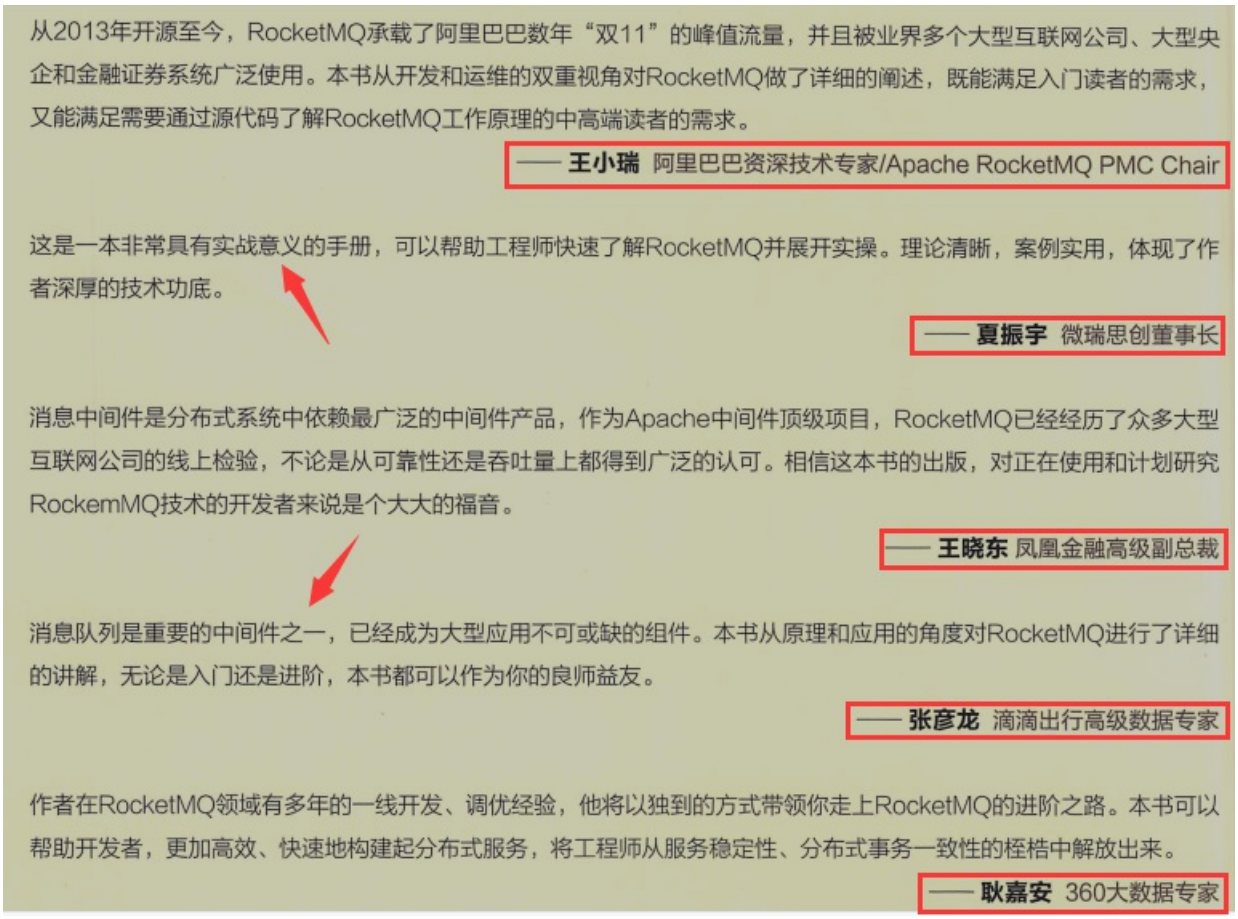
- **强调集群模式无单点，可扩展，任意一点高可用，水平扩展**
- **海量数据的堆积能力，消息堆积后，写入延迟低**
- **支持上万个队列**
- **消息失败重试机制**
- **消息可查询**
- **开源社区灵活**
- **成熟度（支持阿里双十一）**

RocketMQ 核心的四大组件：Name Server、Broker、Producer、Consumer 每个组件都可以部署成集群模式进行水平扩展。

如果搞懂这份RocketMQ实战与原理解析，在简历上写精通RocketMQ完全没问题

由于篇幅原因，这份RocketMQ核心实战原理解析已经被整理成了PDF文档，有需要这份RocketMQ核心实战原理解析完整文档的麻烦转发后私信回复“666”即可获取资料免费领取方式！

阿里、滴滴等资深专家对RocketMQ实战与原理解析的评价



> 注：需要RocketMQ核心笔记【点击此处即可免费领取】

RocketMQ实战目录展示

RocketMQ实战原理解析从入门到生产环境、分布式消息列队、可靠性、吞吐量、Apache中间件、NameServer源码、主从同步，最后基于Netty的通信实现

1、快速入门

本篇幅可以让你了解RocketMQ和分布式消息队列的功能，然后搭建好单机版的消息队列，进而能够发送并接收简单的消息。

- **消息队列功能介绍**

- **RocketMQ简介**
- **快速上手RocketMQ**

阿里的消息中间件有很长的历史，从 2007 年的 Notify 到 2010 年的 Napoli，2011 年升级后改为 MetaQ，然后到 2012 年开始做 RocketMQ，RocketMQ 使用 Java 语言开发，于 2016 年开源。第一代的 Notify 主要使用了推模型，解决了事务消息；第二代的 MetaQ 主要使用了拉模型，解决了顺序消息和海量堆积的问题。RocketMQ 基于长轮询的拉取方式，兼有两者的优点。

每一次产品迭代，都吸取了之前的经验教训，目前 RocketMQ 已经成为 Apache 顶级项目。在阿里内部，RocketMQ 很好地服务了集团大大小小上千个应用，在每年的双十一当天，更有不可思议的万亿级消息通过 RocketMQ 流转（在 2017 年的双十一当天，整个阿里巴巴集团通过 RocketMQ 流转的线上消息达到了万亿级，峰值 TPS 达到 5600 万），在阿里大中台策略上发挥着举足轻重的作用。

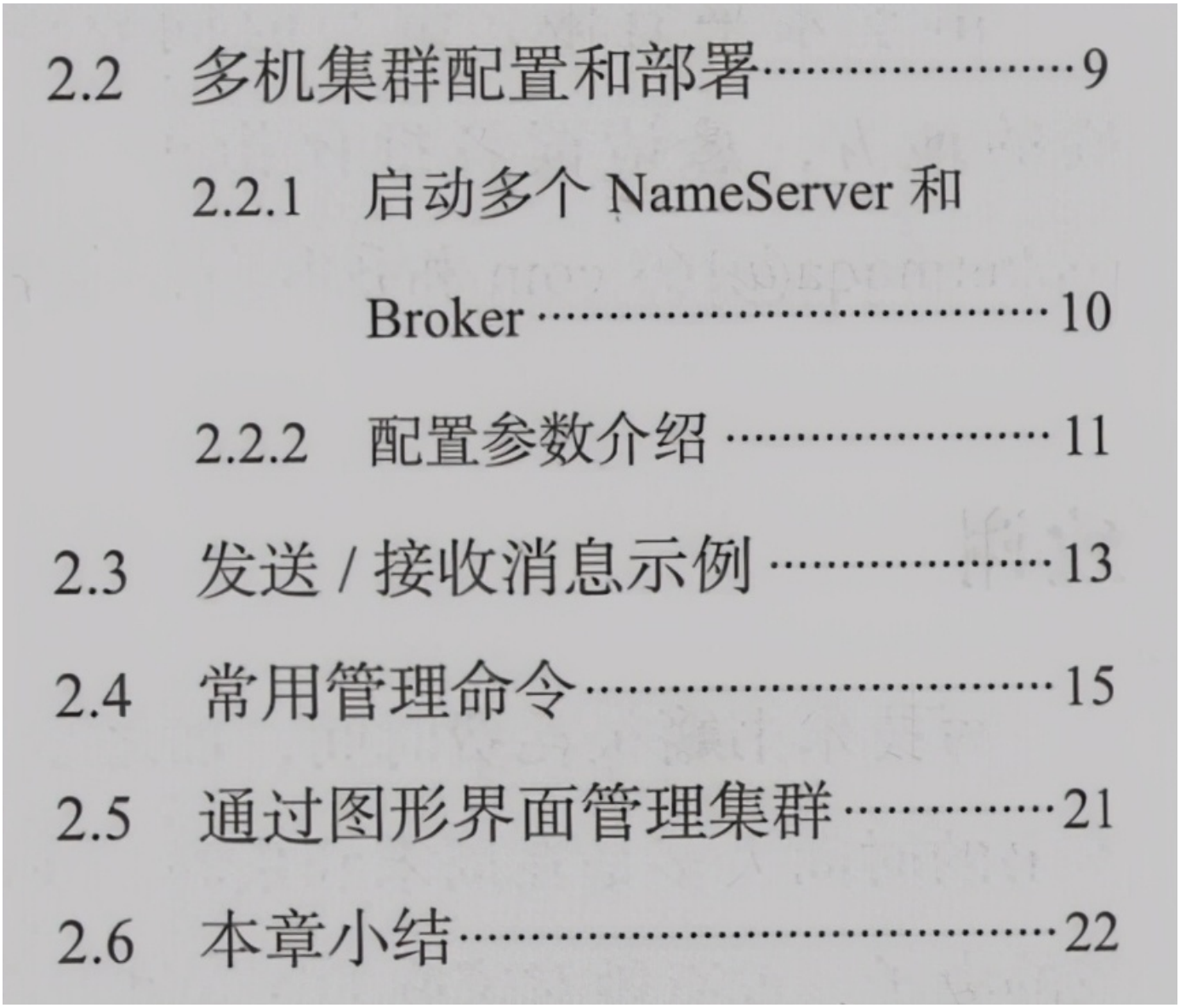
此外，RocketMQ 是使用 Java 语言开发的，比起 Kafka 的 Scala 语言和 RabbitMQ 的 Erlang 语言，更容易找到技术人员进行定制开发。

2、生产环境下的配置和使用

在生产环境中使用RocketMQ集群需要比QuickStart部分了解更多的内容，本章节在机器角色、集群配置和部署，以及集群管理方面都做了介绍，用户可以基于这些内容搭建起一个生成环境的RocketMQ消息队列集群，在数据量不大的非关键场景，可以通过这一章节快速上线。

- **RocketMQ各部分角色介绍**

- **多机集群配置和部署**
- **发送、接收消息示例**
- **常用管理命令**
- **通过图形界面管理集群**



3、用适合的方式发送和接收消息

对消息队列使用者来说，Consumer和Producer是打交道最多的两个类型。

本篇详细介绍了两种类型的Consumer和一种类型的Producer,用户在使用的时候基于业务需求来选择合适的类型。最后重点介绍了Offset和Log，了解Offset机制是正确使用RocketMQ的基础，合理使用Log可以大幅提高开发、调试的效率。

- **不同类型的消费者**
- **不同类型的生产者**
- **如何存储队列位置信息**
- **自定义日志输出**

4、分布式消息队列的协调者

介绍了NameServer的功能，NameServer在RocketMQ集群中扮演调度中心的角色。

各个Producer、Consumer 上报自己的状态上去，同时从NameServer获取其他角色的状态信息。NameServer的功能虽然非常重要，但是被设计得很轻量级，代码量少并且几乎无磁盘存储，所有的功能都通过内存高效完成。

> 还介绍了底层的通信机制，RocketMQ基于Netty对底层通信做了很好的抽象，使得通信功能逻辑清晰，代码简单。

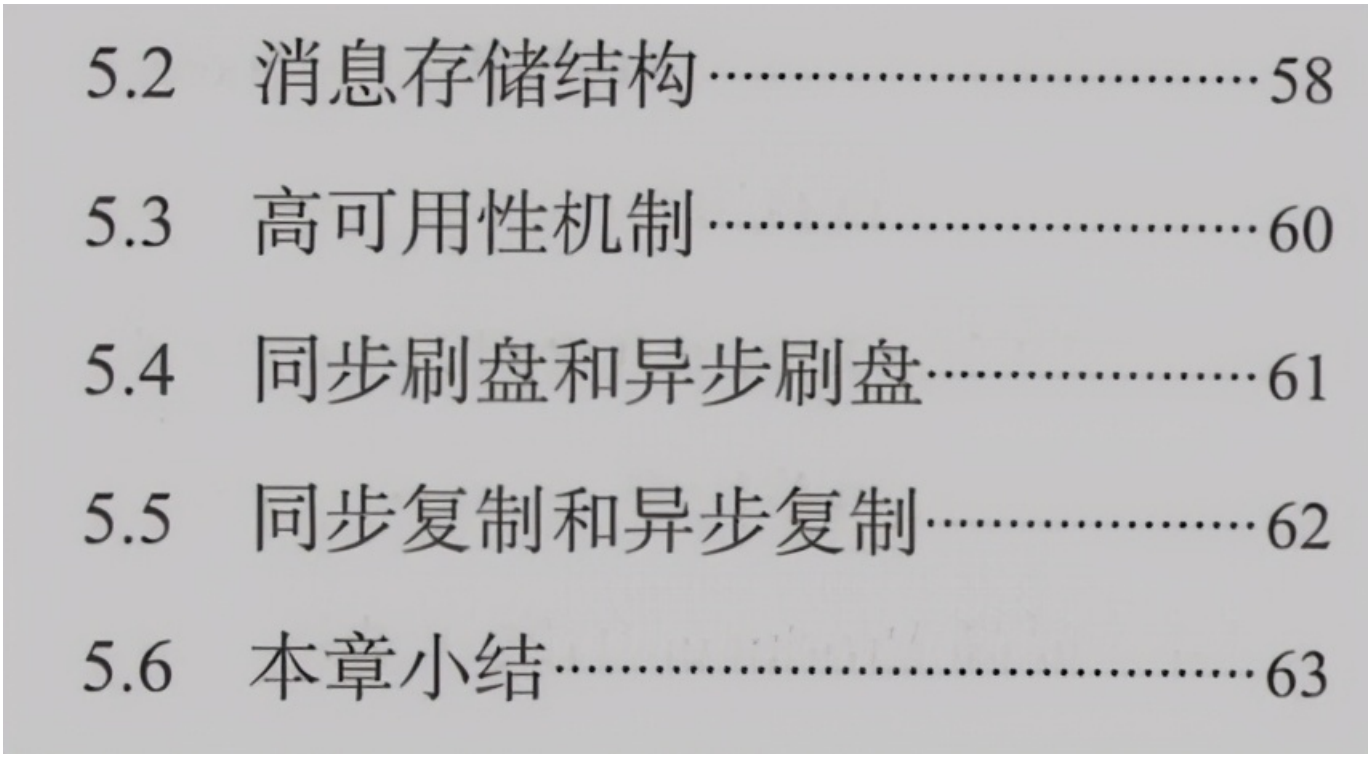
- **NameServer的功能**
- **各个角色间的交互流程**
- **底层通信机制**

5、消息队列的核心机制

介绍了RocketMQ消息队列实现的难点及核心，即“队列”本身的实现，基于磁盘做一个读写效率高的队列并非易事，实现不好就会使磁盘操作成为整个系统的瓶颈，无法提升系统的吞吐量。RocketMQ基于“顺序写”“随机读”的原则来设计，利用“零拷贝”技术，克服了磁盘操作的瓶颈。

> 另一个难点是为了高可用性而设计的主从机制，数据被及时复制到多个机器，这样当一台机器出故障后，整体系统依然可用。这样可靠性和性能能直接有个权衡，RocketMQ 把选择权留给用户，用户根据具体的业务场景来选择要更高的可靠性，还是要更高的效率。

- **消息存储和发送**
- **消息存储结构**
- **高可用性机制**
- **同步刷盘和异步刷盘**
- **同步复制和异步复制**



6、可靠性优先的使用场景

本篇根据使用场景，讨论如何“可靠”地收发消息。即在要求消息顺序的场景下，如何既能并发执行，又能保证消息顺序；然后分析可能的故障场景下，如何应对以保证不丢消息、 不中断服务。RocketMQ 在设计上，有重试机制来保证消息不丢，造成的结果是可能存在消息重复，这一点需要用户根据具体业务场景来处理。下一章将讨论处理大数据量消息的方法。

- **顺序消息**
- **消息重复问题**
- **动态增减机器**
- **各种故障对消息的影响**
- **消息优先级**

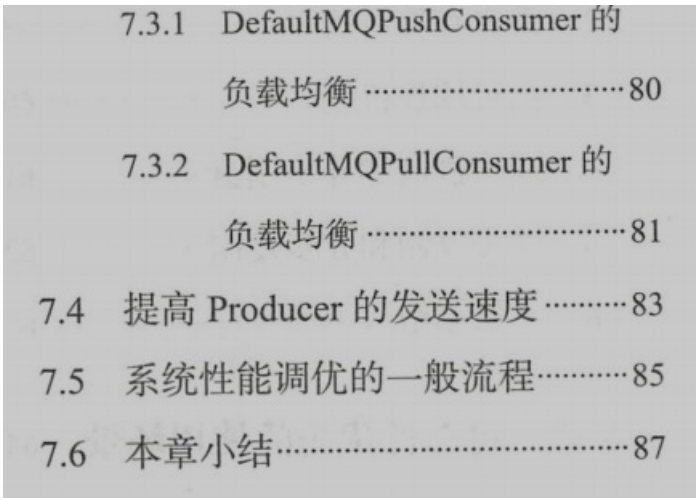
7、吞吐量优先的使用场景

本篇重点关注性能，关注在大消息量的情况下，如何提高RocketMQ的吞吐量。首先介绍了消息过滤，在服务端进行消息过滤可以减少无效消息传输造成的带宽浪费，Tag是最常用的一种高效过滤方式，此外还可以用SQL表达式、FilterServer来过滤消息。

> 另一个提高吞吐量的方法是增加集群的机器数量，提高并发性，要根据实际场景增加Broker、Consumer 或Producer角色的机器数量。

- **在Broker端进行消息过滤**

- **提高Consumer处理能力**
- **Consumer的负载均衡**
- **提高Producer的发送速度**
- **系统性能调优的一般流程**



> 由于篇幅原因，这份RocketMQ核心实战原理解析已经被整理成了PDF文档，有需要这份RocketMQ核心实战原理解析完整文档的麻烦转发后私信回复“666”即可获取资料免费领取方式!

8、和其他系统交互

作为一个中间件产品，会比普通软件更多地需要和其他系统打交道，本篇介绍了如何与SpringBoot、 Spark、Flink等软件进行交互。同时介绍了使用云端的RocketMQ产品，以及自定义开发运维工具的方法。

- **在SpringBoot中使用RocketMQ**
- **直接使用云上RocketMQ**
- **RocketMQ与Spark、Flink对接**
- **自定义开发运维工具**

9、首个Apache中间件顶级项目

RocketMQ是阿里最优秀的中间件之一，本篇介绍了RocketMQ的历史，以及其目前作为Apache顶级项目的现状。

- **RocketMQ的前世今生**
- **Apache顶级项目(TLP)之路**

- **源码结构**
- **不断迭代的代码**

2016 年，MetaQ 在双十一期间承载了万亿级消息的流转，跨越了一个新的里程碑，同时 RocketMQ 进入 Apache 孵化。

图 9-1　RocketMQ 演进历史

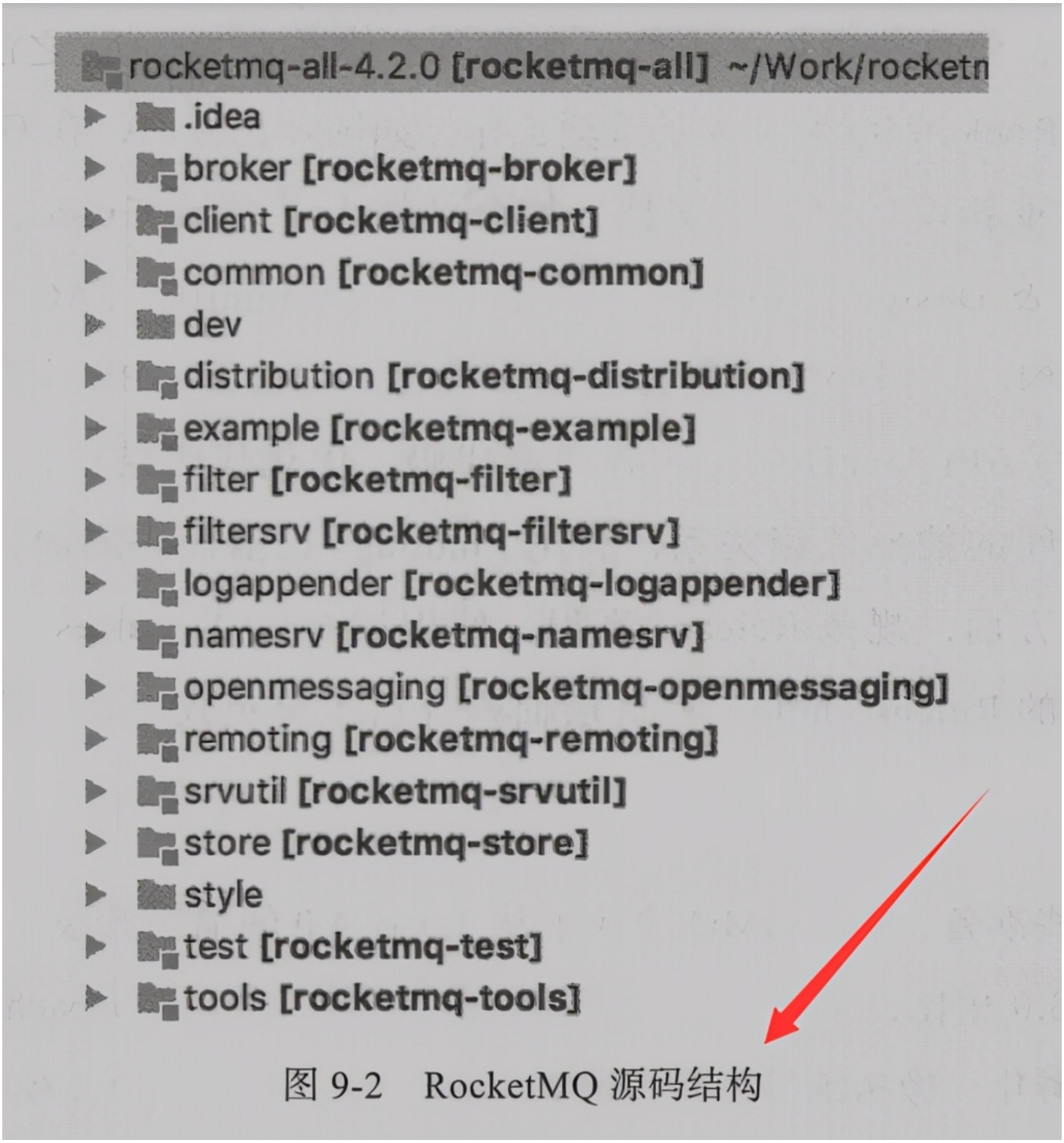


图 9-2　RocketMQ 源码结构

10、NameServer源码解析

本篇分析了NameServer模块的源码，NameServer是--个功能重要但是代码量不大的模块，所以选择这个模块入手，比较容易理解。我们在分析源码时，认真读懂一个模块后就可以对作者的代码风格、设计偏好等有基本的了解。

- **模块入口代码的功能**
- **NameServer的总控逻辑**
- **核心业务逻辑处理**
- **集群状态存储**

11、最常用的消费类

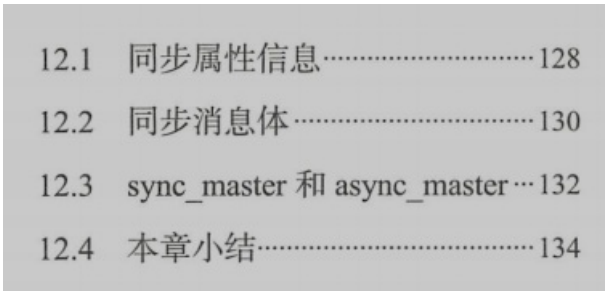
本篇分析的是Client模块里的代码，我们在使用RocketMQ的时候，更多的是和这个模块里的代码打交道。本章重点分析了DefaultMQPushConsumerImpl类，然后分析了Consumer的并发处理过程，最后分析了客户端Class统一的底层通信类MQClientInstance。

- **整体流程**
- **消息的并发处理.**
- **生产者，消费者的底层类**

12、主从同步机制

本篇分析了Master和Slave角色的Broker之间同步信息功能的实现。需要同步的信息分为两种类型，实现方式各不相同：一种是元数据信息，采用基于Netty的command方式来同步消息;另一种是commitLog信息，同步方式是直接基于JavaNIO来实现。

- **同步属性信息**
- **同步消息体**
- **sync_ master 和async_master**



13、基于Netty的通信实现

本篇介绍了RocketMQ底层通信的实现机制，由于它是基于Netty来实现的，所以首先介绍了Netty的基础知识。Netty被用在很多开源软件的底层通信部分，RocketMQ以Netty为基础，还实现了-种机制，把通信功能和消息处理功能分离，不同类型的通信内容被抽象成发送带有对应类型代码的Command，同时根据类型代码查找对应的Processor和Executor来执行，结构非常清晰，为我们自己实现网络通信程序提供了参考。

- **Netty介绍**
- **Netty架构总览**
- **Netty 用法示例**
- **RocketMQ基于Netty的通信**